AN2061
APPLICATION NOTE

EEPROM Emulation with ST10F2xx

Description

External EEPROMs are often used in automotive applications to store adaptative/evolutive data. On the other hand, the Microcontroller used in those systems, are more and more based on embedded-Flash.

The trend to continuously reduce the number of components is forcing designers to look to use Flash memory to emulate EEPROM.

This application note will explain the differences between external EEPROMs and embedded-Flash and will give advises on how to substitute external EEPROM to emulated-EEPROM using the on-chip Flash of ST10F2xx devices.

Although the concept is easy to explain and implement “as is”, there are some embedded aspects that have to be taken into account.

In this application note, the handling of embedded aspects to secure the content of an external EEPROM are assumed to be known by the reader. So, this document is focusing on the differences between EEPROMs and embedded-Flash.

Table Of Contents

1 Introduction 3

2 Embedded-Flash and EEPROM 4

2.1 Difference in write access time 4

2.2 Difference in writing method 4

2.3 Difference in erase time 4

2.4 Additional information on Flash. 5

3 EEPROM emulation concept 6

3.1 Principle 6

3.2 Program/erase cycle 6

3.3 Read-While-Write 7

3.4 Flash organization. 7

3.5 Data-set status bits 7

3.6 Active Flash bank selection 7

4 Embedded application aspects 9

4.1 Reading the Flash while erasing or programming 9

4.1.1 Suspend and resume commands 9

4.1.2 Minimum software to be copied into the on-chip RAM 9

4.2 Data programing / erasing with ST10F2xx 10

4.2.1 Flash field reprogramming 10

4.2.1.1 Completion of the programming process 10

4.2.1.2 Completion of the erasing process 10

4.2.1.3 Safety aspects 10

4.3 Field Reprogramming with ST10F2xx. 11

4.3.1 Field events and Flash reliability 11

4.3.2 List of Events and Suggested Handling Methods 11

4.3.2.1 Reset 11

4.3.2.2 Supply variations. 11

4.3.2.3 Temperature out of specification. 11

4.3.2.4 ST10 PLL Unlock 12

5 Summary 13

6 REVISION HISTORY 14

1 INTRODUCTION

Substituting external EEPROM with emulated EEPROM from the embedded-Flash of the Microcontroller is a complex development. This application note assumes that readers are already familiar with the techniques used to secure the content of evolutive information in external EEPROM of embedded applications.

This application note is organized in 3 parts:

- description of the differences between external EEPROMs and embedded-Flash,
- general description of EEPROM emulation concept,
- introduction to embedded application aspects.

Although this application note is focused and applicable to ST10F269, ST10F280, ST10F276 (and its derivatives: ST10F275, ST10F273, ST10F272, ST10F271), ST10F252 and ST10F296, most of its content is not dependent on the microcontroller.

2 EMBEDDED-FLASH AND EEPROM

Before describing the proposed concept for EEPROM emulation, it is important to remember the main differences between the embedded-Flash memory of a microcontroller and serial external EEPROMs. Those differences are generic to any microcontroller (i.e.: not specific to ST10F2xx variants). They are summarized in the table below.

Table 1. Differences between Embedded Flash and EEPROM

Feature	EEPROM	Emulated EEPROM from embedded-Flash
Write time	some ms random byte : 5 to 10ms page: equivalent to hundred us / word (5 to 10ms per page)	some us (ex : 16us per word)
Erase time	N/A	seconds (ex : 1.5s)
Write method	once started, is not CPU dependent; needs only proper supply.	once started, is CPU dependent: a CPU reset will stop the write process even if supply stays inside specification.
Write access	serial : hundred us random word : 92us page : 22.5us /byte	parallel : hundred ns very few CPU cycles per word.

2.1 Difference in write access time

As Flash has shorter write access time, critical parameters can be stored faster in the emulated EEPROM than in a serial external EEPROM, thereby improving the robustness of the system if the same safety concept is kept.

2.2 Difference in writing method

One of the important differences between external EEPROM and emulated EEPROM for embedded applications is the writing method.

- **Stand-alone external EEPROM:** once started by the CPU, the writing of a word cannot be interrupted by a CPU reset. Only supply failure will interrupt the writing process; so properly sizing the decoupling capacitors can secure the complete writing process inside a stand-alone EEPROM.
- **Emulated EEPROM from an embedded-Flash:** once started by the CPU, the writing can be interrupted by a power failure and by a CPU reset.

This difference should be analysed by system designers to understand the possible impact(s) in their applications and to define the proper handling method.

2.3 Difference in erase time

The difference in erase time is the other important difference between stand-alone EEPROMs and emulated EEPROM with embedded-Flash. Unlike Flash, EEPROM does not require a block erase operation to free-up space before write. This means that some form of software management is required to store data in Flash. Moreover, as the erase process of a block in the Flash takes few seconds, power shut-down and other spurious events that may interrupt

the erase process (ex: reset) should be considered when designing the Flash management software. This means that to design a robust Flash management software it is necessary to have a deep understanding of the Flash erase process.

The Flash erase process is split in 3 phases:

– **phase1:** write all bits to 0, starting from the initial content. Interrupt during this phase will result in more memory cells with a “0” logic level; the content after interrupt in phase1 depends on the Flash initial content.

– **phase2:** write all bits to 1, starting from the all “0” configuration. The longer the time before this phase is interrupted, the higher number of cells will return a “1” logic level. The content after interrupt in this phase does not depend on the Flash initial content; the content after phase2 interrupt, shall be regarded as a totally random content.

– **phase3:** equalization. This phase is necessary to recover over-erased cells. The Flash management software for EEPROM emulation should guarantee that this phase was successfully completed before programming in this bank.

The consequence of interrupt during phase2 is that a single bit approach should be avoided to flag the completion of the erasing process (see more details in Section 3.5 *‘Data-set status bits’* on page 7).

The consequence of interrupt during phase1 and/or phase2 is that it is recommended to have fixed data inside the emulated EEPROM so that checksum can be run to tell which Flash bank keeps the valid data.

The most important point is to ensure that the Flash has been completely erased (phase3 was not interrupted) before programming data inside a bank.

Note: the design of Flash software management is easier if programming in a new bank is always made just after erasing of this bank (when erasing of one bank is necessary).

2.4 Additional information on Flash

Incremental programming: the Flash controller will accept to program a word that is already programmed if the new word is adding more “0” bits.

Programming completion: programming completion is important to guarantee data retention time; the programming is complete when the Flash controller status indicates the end of programming without showing any error flag. If programming is interrupted (ex: supply fail, CPU reset), the cells of the word being programmed will be partially programmed. This can result in unstable “0”s when reading this word.

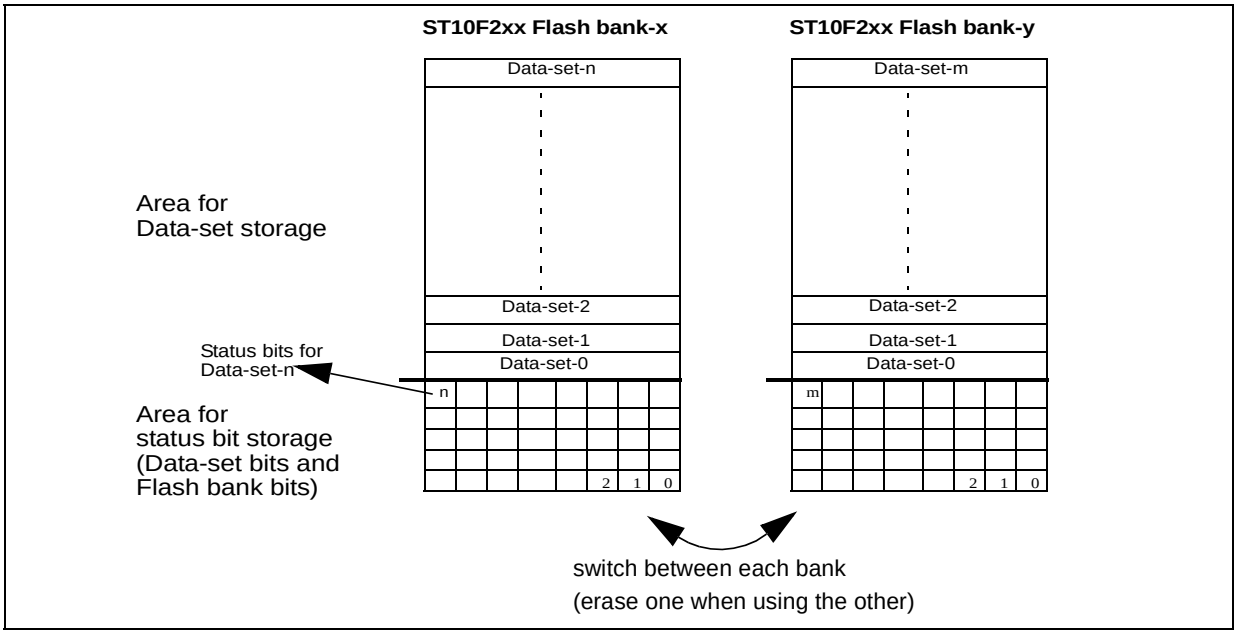
3 EEPROM EMULATION CONCEPT

3.1 Principle

Different concepts are described in the literature. Each of them rely on partitioning a bank of the Flash into several Data-sets and on using control bits (per Flash bank and per Data-set) to compute which Flash bank and which Data-set is the valid one. For variable-length Data-sets, linked data-list structures should be considered.

The method described in this document is based on fixed-length Data-sets and 2 Flash banks.

Figure 1. Bank partitioning for emulated EEPROM



Variable update frequency

The variable update frequency may set high requirements on the program / erase cycle of the Flash and on the Flash features (ex: Read-While-Write). Those features are analysed in details in the following pages.

3.2 Program/erase cycle

The requirements on program / erase cycles are computed by dividing the needed number of erase cycles by the total number of Data-sets in the Flash banks (example with Figure 1: n+m). When this number is still higher than the Flash write/erase endurance characteristics, a closer analysis is needed to understand when the Data-sets are updated:

– when Data-sets need to be updated during operation, it is proposed to use a buffer in RAM and to save the data before shutting-down the microcontroller.

– when Data-sets are updated only before power-down sequence, it is proposed to increase the size of the Flash bank or to use a 3rd bank (see additional information in Chapter 4).

3.3 Read-While-Write

Most currently available Flash technologies must complete a program or erase operation before code or data can be read from another memory block. There is a common misconception that EEPROM emulation can only be done when Read-While-Write functionality is implemented. Read-While-Write, when present, allows to access other memory blocks during erasing and programming; this means that the CPU program does not need to be copied into RAM during programing / erasing. Read-While-Write does not prevent to have a RAM buffer if access into the emulated-EEPROM is needed during programming.

When Read-While-Write is not supported, program or erase suspend command can be used to temporarily read code. All ST10F2xx variants support program and erase suspend commands.

3.4 Flash organization

The concept described above is showing the Flash bank split in 2 parts:

– Data-set storage: keep all variable information,

– Status bit storage: keep status of the Flash bank and of the Data-sets.

Other organizations are possible (ex: to include the Data-set status bits inside each Data-set) for which users may see advantages for their application.

3.5 Data-set status bits

Two status bits are proposed for each Data-set with the combinations shown in the following table.

Table 2. Status bit for Data-set

Status bit value	Meaning
11	Data-set without data (virgin)
01	Data-set with valid data (programmed)
00	Data-set with old or invalid data (dirty)
10	Reserved (invalid)

With those combinations, the status bits will change from “11”, combination after erase, through “01” till “00”, configurations reached by incremental bit programming. Reserved configuration should be used by user’s software to detect which Flash bank hold valid data.

User’s software should define rules on handling Data-set status bits so that in any possible situation in the application (ex: CPU reset), the software can retrieve which bank is active and which Data-set has the valid information.

Ex: when the data have been successfully written into the new Data-set, the status bits of the new and old Data-sets must be updated. A specific sequence to achieve these updates should be defined.

3.6 Active Flash bank selection

The detection of the active Flash bank after power-up should rely on the analysis of the information inside the banks. To support this analysis, specific status bits in each Flash bank can

be used to specify when the relevant data is on the other Flash bank. The following table is showing an example of status bits that can be used for this purpose.

Table 3. Status bit for Flash banks

Status bit value	Meaning
01111111	current bank is active
00111111	erase of the other bank is started
00011111	completion of the erase of the other bank
00001111	new data programming in the other bank is started
00000111	new data programming in the other bank is completed
00000011	the other bank is active

To improve the detection in case of partially erased bank and as explained in Chapter 2, it is proposed to insert a small number of fixed data inside the Data-sets so that by running check-sum it is possible to detect if a given Flash bank has valid or invalid data.
Then, the EEPROM emulation software should analyse the content of the Flash banks detected to contain valid data, in order to check the consistency of the status bits. This algorithm is application dependent as the possible combinations depend on the selected implementation and the different events that the application has to withstand due to the design features (ex: power-fail).

4 EMBEDDED APPLICATION ASPECTS

This chapter is giving advises for embedded applications where the ST10 embedded-Flash memory is the only non volatile memory available.

4.1 Reading the Flash while erasing or programming

Depending on which ST10F2xx variant is used, the entire Flash may not be visible to the CPU when a Flash bank is erased or programmed. When a given ST10F2xx does not support Read-While-Write, the EEPROM emulation software should:

– **disable all interrupts:** as during erasing and programming, it is not possible to have access to ST10 interrupt vector table;

– **copy into the RAM few routines:** before erasing and programming, few software routines shall be copied from the Flash into the on-chip RAM; this should include at least the routines to generate erasing and programming and the routines waiting for the end of erasing or programming;

– **temporarily disable code protection:** if code protection is activated, it should be temporary disabled before executing the code copied into the on-chip RAM.

To cope with application constraints, users may be obliged to support communication during the programming / erasing. This requires that:

– more software is copied into the on-chip RAM,

– the software loop polling the Flash status register is modified to handle the communication process,

– the minimum communication handler (used during polling) is not using interrupts,

– program / erase suspend command is used.

4.1.1 Suspend and resume commands

As stand-alone Flash memories, ST10F2xx embedded-Flash controller supports suspend and resume commands; this allows to suspend at any time the erasing or the programming process and resume it later on.

Once suspend command is completed, ST10F2xx can access to the software routines that may be needed (ex: communication driver) and that were not relocated into the on-chip RAM.

Note: the total time for which the Flash is not available is unchanged but this gives the possibility to suspend the process to run specific routines during Flash erasing or programming (ex: communication protocol).

4.1.2 Minimum software to be copied into the on-chip RAM

The minimum software to be copied into the on-chip RAM is:

– functions issuing the erasing and or programming commands to the ST10 embedded-Flash controller,

– functions polling the Flash status register for detecting the completion of the command and for error detection (see detailed specification),

– watchdog refresh (if activated).

4.2 Data programing / erasing with ST10F2xx

Flash programing / erasing in the field, requires to be able to deal safely with all the possible events that may occur in the application.

This analysis is application dependent and has to be carefully conducted by the user. This section assumes that users have experience with all the generic aspects of field reprogramming; this section will focus only on ST10 specific events.

4.2.1 Flash field reprogramming

This paragraph is giving few advises for the field reprogramming of data. Those advice are not specific to ST10; they are generic to any embedded application that reprograms itself a Data-set.

The main points to control during Flash programing / erasing are:

– completion of the programing / erasing process itself,

– events that may interrupt the reprogramming process.

4.2.1.1 Completion of the programming process

The programming process is completed when the last word to be programmed has been programmed correctly (i.e.: status returned by the Flash is OK). Usually, the last word programmed is an update of the status word (or status bits) of the new Data-set.
If, for any reason, the programming process is interrupted, at the next restart, the value read may be either erroneous or good but with a limited retention time.
This should influence how users are coding the status bits and how supply failures are detected (early warning) or prevented (CPU controlled voltage regulator). It is here difficult to be specific as choices will depend on application requirements and constraints.

4.2.1.2 Completion of the erasing process

As explained in Section 2.3, the completion of the erasing process before programming in a bank is very important. Single bit information to record successful erase process should be avoided. Whenever possible:

– the erase (if necessary because one Flash bank is full) should be done just before Flash programming

– the Flash programming should start only after the successful completion of the erasing process.

As the erase process can take few seconds, this may require to use software controlled voltage regulators to allow to erase the Flash after the main system is stopped (ex: ignition key is removed).

4.2.1.3 Safety aspects

Depending on safety constraints, the usage of a 3rd bank may be considered: instead of using 2 banks alternatively, 3 banks are used alternatively; in the event of a failure (hardware or non-recoverable software error) inside a bank, there are still 2 banks available. Such a technique is already used for non-automotive applications for EEPROM emulation using stand-alone Flash memories.

4.3 Field Reprogramming with ST10F2xx

Flash programming/reprogramming in the field, requires to be able to deal safely with all the possible events that may occur in the application.

This analysis is application dependent and has to be carefully conducted by the user. This section assumes that users have experience with all the generic aspects of field reprogramming and will focus only on ST10F2xx specific events.

4.3.1 Field events and Flash reliability

User's applications must meet ST's recommendations for Flash programming and erasing. Failure to do so, could result in lower data retention and/or altered Flash reliability. The conditions leading to an altered data retention or to an altered reliability depend on the command issued to the Flash and the event that occurred during this command (supply out of range, reset).

From FMEA's perspective, customers should consider that when ST advice are not implemented, Flash reliability can be altered. When ST advice are implemented and provided all field specific events are within ST recommendations (see hereafter), the Flash will meet ST published specification.

4.3.2 List of Events and Suggested Handling Methods

4.3.2.1 Reset

Reset is one of the events possible during field reprogramming, whatever the possible causes of reset (spurious reset, external hardware reset, reset due to power-shut down).

Detection Method:

Reset can occur at any time and there is no possibility to prevent this.

Suggested Handling Method:

Restart the Flash command that was interrupted (i.e.: erasing or programming); use status bits and Flash information to recognize this event.

4.3.2.2 Supply variations

ST10 supply must be kept within the limits published in the Data Sheet during any erase or programming command.

Detection method:

A specific hardware should be added to monitor the supply and reset the ST10F2xx device when the supply is going out of the functional specification.

Suggested Handling Method:

Restart the whole Flash command (i.e.: erasing or programming).

Note: As for any other parameter, the ST10 supply should stay within the maximum absolute ratings defined in the published Data Sheet.

4.3.2.3 Temperature out of specification

Temperature during erasing, programming and read / fetch operations is influencing the reliability of the embedded-Flash.

The embedded-Flash must be programmed and erased only while the junction temperature is

within the limits published in the Data Sheet (see relevant product documentation).

Failure to do so, could result in degraded reliability (lower number of erase cycles, lower data retention).

4.3.2.4 ST10 PLL Unlock

As the Flash programing / erasing timings are not defined by the ST10 CPU core, PLL unlock has no effect on the Flash erasing and programming. Usually, PLL unlock will stop communication because of change in bit/baud rate.

Detection Method:

Not necessary from ST10 point of view (to be checked with application specific constraints).

Suggested Handling Method:

Application dependent.

5 SUMMARY

This application note has shown that by careful identification of events that can happen in the field, and by definition of the Flash organization and its associated control bits, it is possible to define a method to substitute external EEPROM with the embedded-Flash of a microcontroller.

Embedded aspects, handling of the different events that can happen in the field and the needed safety level are the key factors that should influence the emulation concept described in this application note.

6 REVISION HISTORY

Table 4. Revision History

Date	Revision	Description of Changes
November 2004	1	First Issue

The present note which is for guidance only, aims at providing customers with information regarding their products in order for them to save time. As a result, STMicroelectronics shall not be held liable for any direct, indirect or consequential damages with respect to any claims arising from the content of such a note and/or the use made by customers of the information contained herein in connection with their products.

Information furnished is believed to be accurate and reliable. However, STMicroelectronics assumes no responsibility for the consequences of use of such information nor for any infringement of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of STMicroelectronics. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all information previously supplied. STMicroelectronics products are not authorized for use as critical components in life support devices or systems without express written approval of STMicroelectronics.

The ST logo is a registered trademark of STMicroelectronics.
All other names are the property of their respective owners

© 2004 STMicroelectronics - All rights reserved

STMicroelectronics group of companies
Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America
www.st.com